Informatique / iut de BORDEAUX

S1 – modules M1102 et M1103

Algorithmique et Programmation : TD

Calendrier

Ce tableau présente, pour chaque semaine du semestre, quels exercices sont traités, et quelles notions sont censées être maîtrisées. Cela inclut aussi les notions présentées lors des projets. Les points techniques abordés lors des TP (java, git, netbeans, etc.) sont recensés dans le dossier `TP` sur `Bibliotheque`.

Semaine	Exo TD de	à	Notions (et *concepts objet*)
1	–	–	
2	1.1	1.2	variable et type, trace
3	1.3	1.5	syntaxe ExAlgo, action et fonction, boucles Tant Que et Pour
4	1.6	1.8	Selon Que et `switch`, *classe, instance, objet, attribut, passage de référence*
5	1.9	2.1	boucle Répéter-TantQue, *constructeurs, attributs de type objet, tableaux d'objets*
6	2.2	2.8	tableau (déclaration, passage en paramètre)
7	2.9	2.11	ajout / suppression / recherche dans un tableau, *méthodes*
8			*méthodes (de classe / d'instance), surcharge, énumérations, égalité*
9	3.1	3.5	recherche dichotomique, tableaux à 2 dimensions et boucles imbriquées
10	4.1	4.5	chaînes de caractère
11	5.1	5.5	fichiers et accès séquentiel
12	6.1	6.5	récursivité
13	7.1	7.2	tri par sélection, tri à bulle
14	7.3	7.5	tri par insertion, tri rapide, tri fusion
15	8.1	8.2	conteneurs : tableau extensible, tableau associatif
16			conteneurs : ensemble

Table des matières

Semaine 1

Introduction à Processing : sur machine, pas de TD.

Semaine 2

1 Structures de contrôle

1.1 Lecture de code Processing

Rappelons que l'instruction `random(v)` renvoie une valeur aléatoire de type `float` comprise entre `0` inclus et `v` exclu. Ainsi :

— `random(21)` renvoie un `float` dans $[0, 21[$.
— `(int)random(21)` renvoie un `int` dans $[0, 20]$.
— `(int)random(21)-10` renvoie un `int` dans $[-10, 10]$.

Question 1
Que fait le programme suivant ?

```
int nb = (int)random(21) - 10;
while (nb != 0)
{
  println("Tirage = ", nb);
  nb = (int)random(21) - 10;
}
println("Fini");
```

Question 2
Le programme a été modifié. Combien vaut `cpt` à la fin de l'exécution, si l'on considère que le tirage aléatoire a produit les nombres suivants : 4,5,-2,9,0.

```
int cpt = 0;
int nb = (int)random(21) - 10;
while (nb != 0)
{
  println("Tirage = ", nb);
  cpt++;
  nb = (int)random(21) - 10;
}
println(cpt);
println("Fini");
```

Question 3
Le programme a été modifié. Combien vaut `val` si l'on considère que le tirage aléatoire a produit les nombres suivants : 4,5,-2,9,0.

```
int val = 0;
int nb = (int)random(21) - 10;
while (nb != 0)
{
  println("Tirage = ", nb);
  val += nb;
  nb = (int)random(21) - 10;
}
println(val);
println("Fini");
```

Question 4
Le programme a été modifié. Combien vaut `avg` si l'on considère que le tirage aléatoire a produit les nombres suivants : 4,5,-2,9,0.

```
int val = 0;
int nbTirages = 0;
float avg = 0.0;
int nb = (int)random(21) - 10;
while (nb != 0)
{
  println("Tirage = ", nb);
  nbTirages++;
  val += nb;
  nb = (int)random(21) - 10;
}
println("La somme : ", val);
println("Le nombre de tirages : ", nbTirages);
avg = (float)val / nbTirages;
println(avg);
println("Fini");
```

```
if (nbTirages != 0)
{
  avg = (float)val / nbTirages;
  println(avg);
}
else
{
  println("Moyenne non calculable");
}
```

Question 5
Que fait le programme suivant si l'on considère que le tirage aléatoire a produit les nombres suivants : 4,5,-2,9,0.

```
int gap = 0;
int nb = (int)random(21) - 10;
while (nb != 0)
{
  println("Tirage = ", nb);
  if (nb > gap)
  {
    gap = nb;
  }
  nb = (int)random(21) - 10;
}
println(gap);
println("Fini");
```

Question 6
Dans le code précédent, si tous les nombres sont inférieurs à 0, la valeur affichée sera 0. Comment modifier l'initialisation de gap pour éviter cela ?

1.2 Introduction à l'ExAlgo

Nous utilisons dans ce TD un langage nommé ExAlgo, qui permet d'écrire proprement les algorithmes, et de manière plus souple qu'un langage de programmation. Ce langage est uniquement utilisé en interne (au département) pour les questions algorithmiques. ExAlgo est malgré tout assez proche de Processing (et Java) : il est donc facile de passer d'ExAlgo à Processing. Dans cette optique les indices des tableaux commencent à zéro.

Le tableau suivant est **à connaître par cœur** :

Processing	ExAlgo
`int, float, char, boolean`	entier, réel, caractère, booléen
`int a, b;`	**Var:** a,b : entier
`a = 3;`	a ← 3
`a == 3`	a = 3
`print(a); ou println(a);`	Écrire(a)

`if (a == 3) {`
`  ...`
`} else {`
`  ...`
`}`
	Si *a = 3* **Alors**
	└ ...
	Sinon
	└ ...

`for (int i=0; i<10; i++)`
`{`
`  ...`
`}`
	Pour *i* ***de*** *0* **à** *9* **Faire**
	└ ...

`i = 0;`
`while (i<10)`
`{`
`  ...`
`  i++;`
`}`
	i ← 0
	Tant Que *i < 10* **Faire**
	│ ...
	└ i ← i+1

`float maFonction(int a, int b)`
`{`
`  ...`
`  return m;`
`}`
	Fonction *maFonction(*E *a : entier,* E *b : entier) : réel*
	Début
	│ ...
	└ **Retourner** *m*

`void monAction(int a, int b)`
`{`
`  ...`
`}`
	Action *monAction(*E *a : entier,* E *b : entier)*
	Début
	└ ...

Dans une boucle **Pour**, le *pas* par défaut est de 1. Lorsqu'il est différent de 1, il faut le préciser :
Pour *i* ***de*** *9* **à** *0* ***par pas de*** -1 **Faire**
└ ...

Ainsi, on n'entre jamais dans la boucle suivante : **Pour** *i* ***de*** *9* **à** *0* **Faire**
└ ...

De plus, en sortant d'une boucle Pour, l'indice utilisé a une valeur l'ayant fait sortir de l'intervalle. Ainsi, après : **Pour** *i* ***de*** *0* **à** *9* **Faire**
└ ...
i vaut 10 (comme en Processing et Java).

Chaque argument (d'une fonction ou d'une action) est passé :
— en entrée (E) si, dans la fonction ou l'action, il est lu mais pas modifié,
— en sortie (S) si, dans la fonction ou l'action, il est modifié mais pas lu,
— en entrée/sortie (ES) si, dans la fonction ou l'action, il est lu et modifié.

Remarque

En ExAlgo, on dispose de la fonction

```
entierAléatoire(E vInf : entier, E vSup : entier) : entier
```

permettant de générer des *entiers* aléatoirement entre `vInf` (inclus) et `vSup` (exclus). En Processing on réalise cela via l'expression :

```
n = (int)random(vSup - vInf) + vInf;
```

pour garder une distribution uniforme dans les cas où `vInf` et/ou `vSup` seraient négatifs.

Question 1
Écrire en ExAlgo un programme :
- qui tire aléatoirement un entier et l'affecte dans une première variable,
- qui tire aléatoirement un deuxième entier et l'affecte dans une deuxième variable,
- qui enfin échange le contenu de ces deux variables.

Question 2 [facultatif]
Écrire en ExAlgo un programme qui tire aléatoirement un nombre n entre 1 et 100, puis tire aléatoirement (et affiche) n valeurs de températures (entre -10 et $+40$).

Question 3 [facultatif]
Modifier le programme précédent pour afficher la température minimale, la température maximale et la moyenne des températures.

Semaine 3

1.3 De Tant Que à Pour

Pour toutes les questions de cet exercice, avant de fournir le programme en ExAlgo, vous préciserez :

1. s'il s'agit d'une fonction ou d'une action ?
2. quel est (ou quels sont) le(s) paramètre(s) d'entrée ?
3. quel est le type de sortie (éventuellement) ?

Question 1
Écrire une fonction calculant la somme d'une suite d'entiers positifs aléatoires (inférieurs à 100) terminée par zéro.

Question 2
Écrire une action principale associée.

Question 3
En utilisant une boucle `Tant Que`, écrire une fonction qui, étant donné un entier n, tire aléatoirement n entiers entre 0 et 100 et retourne la somme de ces entiers.

Question 4
Écrire une action principale associée.

Question 5
En utilisant une boucle `Pour`, écrire une fonction qui, étant donné un entier n, tire aléatoirement n entiers entre 0 et 100 et retourne la somme de ces entiers.

1.4 Affichage des allumettes [facultatif]

Question 1
Écrire une action qui étant donné un entier n affiche n fois le symbole "|".

Question 2
Écrire une action principale associée.

1.5 Puissances

Question 1
Écrire une fonction puissance qui, étant donnés deux entiers strictement positifs x et n, renvoie x^n. On ne dispose que de l'addition et de la multiplication.

Question 2
Écrire une action principale associée.

Question 3
Combien de multiplications sont réalisées lors du calcul de la puissance ?

Question 4
Écrire une fonction qui étant donnés x et n calcule

$$1 + x + x^2 + x^3 + \cdots + x^n$$

Question 5
Écrire une action principale associée.

Question 6
Combien de multiplications sont réalisées lors du calcul de la puissance ?

Question 7
Peut-on faire moins ? Petit rappel : schéma de Hörner

— pour $n = 3$:

$$1 + x + x^2 + x^3 = 1 + x(1 + x(1 + x))$$

— pour n quelconque :

$$1 + x + x^2 + x^3 + \cdots + x^n = 1 + x(1 + x(1 + x(\cdots(1 + x))))$$

Semaine 4

Structure conditionnelle à choix multiples

Le `Si-Alors-Sinon` permet d'envisager deux choix différents. Pour permettre plusieurs choix possibles, il faut alors utiliser des `Si-Alors-Sinon` imbriqués. Certains langages de programmation offrent une solution plus simple. Voici un exemple :

```
Action mention()
Var: note : réel
Début
    lire(note)
    Selon Que
        note ≥ 16 :
            écrire("très bien")
        note ≥ 14 :
            écrire("bien")
        note ≥ 12 :
            écrire("assez bien")
        note ≥ 10 :
            écrire("passable")
        Sinon :
            écrire("ajourné")
```

Selon Que : Les tests sont effectués dans l'ordre d'apparition des cas. Dès qu'une condition est vérifiée, l'action associée est alors exécutée puis on sort du `Selon Que` sans envisager les cas suivants.

1.6 Cherchez l'erreur

Question 1

Que se passe-t-il avec l'algorithme suivant :

```
Action mentionBis()
Var: note : réel
Début
    lire(note)
    Selon Que
        note > 10 :
            écrire("passable")
        note > 12 :
            écrire("assez bien")
        note > 14 :
            écrire("bien")
        note > 16 :
            écrire("très bien")
        Sinon :
            écrire("ajourné")
```

Nous verrons que dans certains langages de programmation, la structure équivalente au `Selon Que` ne s'écrit pas de cette manière (par exemple `switch` en Processing ou Java). En ExAlgo, il faut voir cette structure conditionnelle comme une simple réécriture de `Si-Alors-Sinon` imbriqués. C'est ce que montre le prochain exercice.

1.7 `Selon-Que` vs `Si-Alors-Sinon`

Question 1
Réécrivez l'action `mention` en utilisant seulement des `Si-Alors-Sinon`.

Question 2
Dessinez l'arbre d'évaluation des conditions.

1.8 Calculatrice

Écrire un algorithme qui, étant donnés deux nombres et une opération arithmétique, affiche le résultat de l'opération.

Semaine 5

Les boucles `Répéter-TantQue`

Ici la démarche est légèrement différente de celle des boucles `Tant-Que`. Le traitement sera effectué une première fois dans tous les cas, puis tant que la condition d'arrêt est vérifiée. Voici comment l'algorithme du premier exercice peut se réécrire :

```
Action ecrireSomme()
Var: n, somme : entier
Début
    somme ← 0
    Répéter
        n ← entierAléatoire(0,100)
        somme ← somme + n
    Tant Que n ≠ 0;
    écrire(somme)
```

La syntaxe est la suivante en ExAlgo :

```
Répéter <action> Tant Que <condition>
```

et en Java :

```
do { <action> } while ( <condition> );
```

1.9 Évaluer la soustraction

On souhaite écrire un programme qui teste, sous forme de jeu, si un enfant sait soustraire. Une partie se déroule ainsi :

- le programme tire au hasard deux entiers entre 0 et 10 et les affiche
- puis il demande à l'enfant quelle en est la différence ;
- l'enfant saisit alors sa réponse ;
- et on répète ainsi les trois étapes précédentes tant que l'enfant n'a pas trouvé la bonne réponse (on tire à nouveau deux entiers au hasard entre chaque tentative).

Le score d'une partie est le nombre de tentatives pour trouver la bonne réponse (au moins 1). Le but pour l'enfant est bien sûr d'obtenir le score le plus bas possible. On suppose que l'enfant connaît aussi les entiers négatifs, on peut donc lui demander "4 - 7 = ?" par exemple.

Question 1
Écrire un programme qui gère une partie, via une fonction qui renvoie le score.

Question 2
On souhaite que l'enfant rejoue (avec des nombres a, b différents) tant qu'il n'a pas réussi une partie avec un score inférieur à n, ce paramètre n étant variable. Écrire un tel algorithme.

1.10 Traduction [facultatif]

Question 1
Afficher les nb premiers entiers (de 1 à nb) à l'aide d'une boucle `Pour`.

Question 2
Afficher les nb premiers entiers (de 1 à nb) à l'aide d'une boucle `Tant-Que`.

Question 3
Afficher les nb premiers entiers (de 1 à nb) à l'aide d'une boucle `Répéter-TantQue`.

2 Tableaux

Nous avons eu l'occasion avec les structures répétitives de faire des calculs sur des suites d'entiers. Il a fallu à chaque fois concevoir les algorithmes de manière à ce que ces calculs soient effectués au fur et à mesure de la saisie car il était "impossible" de stocker un grand nombre de variables entières. Nous présentons dans ce TD la première véritable structure de données : les *tableaux*.

L'idée est de stocker sous une même entité, un ensemble de données ayant les mêmes caractéristiques, un sens commun... En particulier toutes les cases d'un même tableau sont du même *type*. Nous disposons donc de tableaux de caractères, de tableaux d'entiers, etc. Chaque case du tableau possède un indice (numéro) compris entre 0 et $n-1$, si le tableau est de taille n.

Exemple d'un tableau de 5 caractères :

`'j'`	`'f'`	`'d'`	`'z'`	`'f'`	⇐ valeurs
0	1	2	3	4	⇐ indices

En ExAlgo, nous disposons :
- de `t[i]` pour accéder à la case d'indice `i` du tableau `t`. Par exemple `t[1]` vaut `'f'` dans le tableau `t` ci-dessus.
- du type `tableau d'entiers` pour les tableaux d'entiers (de même pour les autres types : caractères, etc). On peut également préciser la taille du tableau dans le type :
  - `tableau[10] d'entiers`
  - `tableau[n] d'entiers`, la variable `n` contient alors la taille du tableau (elle est déclarée par la même occasion).
- de `t.taille` pour obtenir la taille (nombre de cases) du tableau `t`.

Comme les autres types, les tableaux peuvent être passés en paramètre en entrée (**E**), en sortie (**S**), ou en entrée/sortie (**ES**) en ExAlgo.

2.1 Saisie et affichage d'entiers

On souhaite demander à l'utilisateur de saisir des entiers, et ensuite les afficher. Nous allons décomposer ce problème.

Question 1
Écrire un programme permettant de remplir un tableau (en paramètre) avec des entiers saisis par l'utilisateur.

Question 2
Écrire un programme qui affiche le contenu d'un tableau d'entiers.

Question 3
Écrire un programme qui demande à l'utilisateur de saisir 10 nombres, puis les affiche à l'écran.

Question 4
Comment aurait-on pu faire sans tableau ? Quel est l'avantage d'avoir utilisé un tableau ?

Semaine 6

2.2 Tableau ordonné

Question 1
Modifier la fonction `saisir` suivante pour saisir des entiers dans un tableau, qui est censé être ordonné. On ignorera les nombres qui ne sont pas strictement plus grands que leur prédécesseur. Le tableau est de taille 20, on s'arrêtera une fois qu'il est rempli.

Fonction *saisir() : tableau d'entiers*
Var: tab : tableau[20] d'entiers
  i : entier
Début
  Pour *i* ***de*** *0* **à** *tab.taille-1* **Faire**
    lire(tab[i])
  Retourner tab

2.3 Maximum

Question 1
Écrire une fonction permettant d'obtenir l'élément de valeur maximum d'un tableau d'entiers.

Question 2
Modifiez l'algorithme ci-dessus pour que la fonction ne retourne plus l'élément maximum, mais son **indice** (0 si c'est le premier, 1 si c'est le second, etc).

Question 3
Transformez la fonction précédente en action paramétrée afin d'avoir en paramètre de sortie l'indice de l'élément maximum *et* la valeur de l'élément maximum.

2.4 Remplacer toutes les occurrences [facultatif]

Question 1
Écrire un algorithme permettant de remplacer toutes les occurrences d'un caractère x par y dans un tableau de caractères de taille n.

2.5 Rechercher et remplacer la première occurrence

Question 1
Écrire une fonction permettant de savoir si un entier x est dans un tableau d'entiers de taille n.

Question 2
Écrire une fonction permettant de rechercher et de retourner l'indice de la 1ère occurrence d'un entier x dans un tableau d'entiers de taille n s'il est présent, et -1 sinon.

Question 3
Écrire un algorithme permettant de remplacer la 1ère occurrence d'un entier x par y dans un tableau d'entiers de taille n.

2.6 Échange d'éléments

Question 1
Écrire un algorithme permettant d'échanger les éléments d'indice i et j dans un tableau d'entiers de taille n.

Question 2
Écrire un algorithme permettant d'inverser les éléments d'un tableau d'entiers de taille n.
Par exemple 1 7 2 4 devient 4 2 7 1.

2.7 Éléments d'une suite [facultatif]

Question 1
Écrire un algorithme permettant d'afficher les p premiers éléments de la suite définie par :

$$u_0 = 1 \qquad u_n = \sum_{k=0}^{n-1} u_k.u_{n-1-k}$$

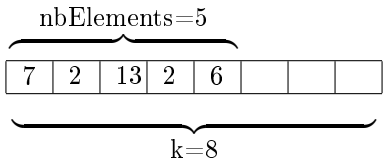
2.8 Nombre d'inversions [facultatif]

Question 1
Écrire un algorithme permettant de calculer le nombre d'inversions (`i < j` et `tab[i] > tab[j]`) dans un tableau d'entiers.

Semaine 7

2.9 Recherche et suppression d'un élément

Dans cet exercice, nous utilisons un tableau pour stocker des entiers. Ce tableau est de taille k. Les éléments sont stockés au début du tableau, et un entier *nbElements* indique le nombre d'éléments "utiles" au début du tableau. Par exemple :



On souhaite établir des algorithmes qui cherchent une valeur dans le tableau d'entiers et, si elle est présente, la suppriment. Si la valeur est présente plusieurs fois, on ne supprime que sa première occurrence.

Question 1
Écrire un tel algorithme dans le cas d'un tableau non trié.

Question 2
Écrire un tel algorithme dans le cas d'un tableau trié.

Question 3
Écrire un tel algorithme dans le cas d'un tableau non trié, dont on souhaite garder l'ordre relatif entre les éléments.

2.10 Ajout d'un élément

On utilise un tableau d'entiers de la même façon que dans l'exercice précédent. On souhaite désormais ajouter un nouvel élément dans un tableau d'entiers. On suppose ici que le tableau n'est pas plein et que l'on peut réaliser cette opération.

Question 1
Écrire un algorithme qui permet d'ajouter un élément, dans un tableau non trié.

Question 2
Même question, mais dans le cas d'un tableau trié.

2.11 Saisie de tableau trié [facultatif]

On souhaite stocker des entiers dans un tableau de la même manière que précédemment.

Question 1
Écrire un algorithme qui permet de saisir un tableau d'entiers en le triant. Entre chaque saisie d'entier on demandera à l'utilisateur s'il souhaite continuer (s'il reste de la place).

Question 2
Écrire l'action principale appelant l'action `saisieTriee`.

Semaine 8

Présentation des classes en Java : diaporama.

Semaine 9

3 Recherche, et révision

3.1 Recherche dans un tableau

Question 1
Proposer un algorithme prenant en entrée un entier et un tableau d'entiers, et retournant l'indice (ou plutôt, *un* indice) où apparaît cet entier dans le tableau. L'algorithme retournera -1 si l'entier n'apparaît pas dans le tableau.

Question 2
On suppose maintenant le tableau trié. Proposer un algorithme permettant de trouver l'indice d'un entier donné, plus performant que dans le cas non trié ci-dessus (penser à une recherche dans le dictionnaire).

3.2 Lecture de code - [Processing] [facultatif]

```
void backToSchool(int[] tab1, int[] tab2, int[] tab3) {
  int taille = tab1.length;
  for (int i = 0; i < taille; i++) {
    tab3[i] = tab1[i] + tab2[i];
  }

  for (int i = taille - 1; i > 0; i--) {
    if (tab3[i] > 9) {
      tab3[i] -= 10;
      tab3[i - 1] += 1;
    }
  }
}

void afficher(int[] tab) {
  int taille = tab.length;
  for (int i = 0; i < taille; i++)
  {
    if (i != 0 && ((taille - i) % 3 == 0)) {
      print(".");
    }
    print(tab[i]);
  }
  println();
}

void setup() {
  final int TMAX = 12;
  int[] tab1 = {0, 9, 8, 0, 5, 8, 3, 1, 7, 2, 9, 3};
  int[] tab2 = {3, 2, 0, 2, 3, 4, 7, 0, 5, 1, 0, 5};
  int[] tab3;
  tab3 = new int[TMAX];
  backToSchool(tab1, tab2, tab3);
  afficher(tab3);
}
```

Question 1
Que donne l'exécution du programme ?

Question 2
Décrire en une phrase ce que fait la fonction `backToSchool`.

Question 3
Que donne l'exécution de `afficher(tab3);` pour :
`tab1 = {9,9,8,0,5,8,3,1,7,2,9,3};`
`tab2 = {3,2,0,2,3,4,7,0,5,1,0,5};`
Expliquez.

3.3 Un peu de calcul - [Java] [facultatif]

Soient les deux fonctions suivantes :

$$f_1(x) = \frac{1}{1-x} \quad \text{et} \quad f_2(x) = \frac{1}{1+x}$$

Question 1
Écrivez les fonctions `fonction1` et `fonction2` de sorte qu'on puisse compiler le `main` suivant :

```
1 public static void main(String[] args) {
2     float x = 0.5f;
3     System.out.println(fonction1(x));
4     System.out.println(fonction2(x));
5 }
```

Question 2
Soient les deux nouvelles fonctions suivantes :

$$g_1(x,n) = 1 + x + x^2 + x^3 + ... + x^n$$

$$g_2(x,n) = 1 - x + x^2 - x^3 + ... + (-1)^n x^n$$

Donnez l'implémentation des fonctions `g1` et `g2`. Ayez la bonne idée de "factoriser" votre code, c'est à dire d'identifier une ou plusieurs fonctions récurrentes paramétrées que vous pourriez utiliser dans chacune de ces deux fonctions. Nous souhaitons maintenant pouvoir compiler ce code :

```
1   float x = 0.5f;
2   int ordre = 5;
3   println( fonction1(x), " approx:", g1(x,ordre));
4   println( fonction2(x), " approx:", g2(x,ordre));
```

Question 3
En fait, la fonction g_1 donne une approximation de f_1 au voisinage de 0, i.e.

$$\lim_{n\to\infty} g_1(x,n) = f_1(x)$$

autrement dit, si x est petit (proche de 0) et que n croit, alors $g_1(x,n)$ s'approche de plus en plus de $f_1(x)$. De manière similaire, si x est petit (proche de 0) et que n croit, alors $g_2(x,n)$ s'approche de plus en plus de $f_2(x)$. Exemples :

$$f_1(0.4) = 1.666 \text{ et } g_1(0,3) = 1 + 0.4 + 0.4^2 + 0.4^3 = 1.624 \quad erreur = 0.042$$
$$f_1(0.4) = 1.666 \text{ et } g_1(0,4) = 1 + 0.4 + 0.4^2 + 0.4^3 + 0.4^4 = 1.6496 \quad erreur = 0.0164$$
$$f_1(0.4) = 1.666 \text{ et } g_1(0,5) = 1 + 0.4 + 0.4^2 + 0.4^3 + 0.4^4 + 0.4^5 = 1.65984 \quad erreur = 0.00616$$

Etant donné ϵ un réel positif, écrire une fonction qui renvoie la plus petite valeur de n telle que :

$$|f_1(x) - g_1(x,n)| < \epsilon$$

3.4 Tableaux - Gestion des doublons - [ExAlgo]

Dans cet exercice on utilise un tableau de taille k pour stocker un certain nombre d'entiers, noté *nbEntiers* dans la suite.

Question 1
Étant donnés un tableau t de *nbEntiers* entiers, et un entier x, écrire une action qui remplace tout doublon d'un élément (à partir de sa deuxième occurrence) par x. Exemple : si t contient 1 2 3 15 15 4 2 26 15 et x égale -1, alors après application de votre action, t contient 1 2 3 15 -1 4 -1 26 -1.

Question 2
Écrire une action qui supprime toutes les occurrences d'un entier x d'un tableau t d'entiers de taille *nbEntiers* et donne la nouvelle taille du tableau par l'intermédiaire d'un paramètre.
Exemple : Si les éléments de t sont 1 2 3 15 -1 4 -1 26 -1 et x égale -1, après application de votre action, t doit contenir les éléments 1 2 3 15 4 26 (potentiellement dans un autre ordre) et la nouvelle taille est 6. Attention à la complexité.

Question 3
Écrire un programme qui, étant donné un tableau t d'entiers positifs de taille *nbEntiers*, supprime tous les doublons et affiche le nombre d'éléments restants.

3.5 Tableaux - Recherche de sous-mots et motifs - [ExAlgo]

Cet exercice a pour but la recherche de sous-mots et de motifs dans un mot donné. Un mot de n lettres peut être codé par un tableau de n caractères où chaque case du tableau contient une lettre.

Exemple : le mot "coucou" peut être vu comme un tableau `tab` de 6 caractères où `tab[0]` contient `'c'`, `tab[1]` contient `'o'`, etc :

tab	c	o	u	c	o	u

Nous définissons un sous-mot `sm` d'un mot `m` comme une séquence de lettres consécutives de `m`. Par exemple, "cou", "ou", "u", "ouco" sont des sous-mots de "coucou", mais "uo" n'en est pas un.

Question 1
Écrire un programme qui, étant donnés un mot `mot` et un sous-mot `sousmot`, renvoie vrai si `sousmot` est un sous-mot de `mot`, et faux sinon.

Nous définissons un motif `mo` d'un mot `m` comme un mot obtenu à partir de `m` en supprimant des lettres. Par exemple, "uc", "uu", "ouu", "c" sont des motifs de "coucou", mais pas "uoo".

Question 2
Écrire un programme qui étant donnés un mot `m` de taille `tm` et un motif `mo` de taille `tmo` renvoie vrai si `mo` est un motif de `m` et faux sinon.

Semaine 10

4 Chaînes de caractères

Dans ce TD, on se propose de manipuler des chaînes de caractères. Nous appelerons ce type : `Chaine` en ExAlgo. Il correspond au type `String` de Java. Les positions d'une chaîne de n lettres vont de 0 à $n-1$.

Une variable de type `Chaine` possède les méthodes suivantes :

`caractère caractèreA(entier index)` : retourne le caractère en position index

`entier positionDe(caractère car, entier index)` : retourne la position du premier caractère `car` trouvé à partir de l'index `index` donné (et `-1` sinon)

`Chaine remplace(caractère oldChar, caractère newChar)` : retourne une nouvelle `Chaine` dans laquelle toutes les occurrences de `oldChar` sont remplacées par `newChar`.

`Chaine remplacePremier(caractère oldChar, caractère newChar)` : retourne une nouvelle `Chaine` dans laquelle la première occurrence de `oldChar` est remplacée par `newChar`.

`Chaine sousChaine(entier beginIndex, entier endIndex)` : retourne une nouvelle chaine extraite de l'originale entre `beginIndex` et `endIndex-1` inclus.

`booléen contient(Chaine ch)` : retourne vrai ssi `ch` est contenu dans la chaine principale.

`entier taille()` : retourne la taille de la chaîne de caractères.

`Chaine concat(Chaine chaine)` : retourne la `Chaine` obtenue en ajoutant celle indiquée en paramètre.

`Chaine ajouterCaractere(caractère car)` : retourne la `Chaine` obtenue en lui ajoutant le caractère `car` à la fin.

4.1 Voyelles

Question 1
Écrire une fonction qui indique si un caractère est une voyelle.

Question 2
Écrire une fonction qui compte le nombre de voyelles contenues dans la chaîne passée en paramètre.

4.2 Extraction et validation de champs

Un programme doit dessiner une carte géographique à l'écran. Pour cela, il utilise un fichier qui indique la position de villes. Chaque ligne du fichier est (en principe) une `Chaine` de la forme

```
Bordeaux,105.2,102.4
```

Le nom de la ville peut contenir des espaces, des tirets et des apostrophes. Les coordonnées sont des nombres entiers ou décimaux.

Question 1
Sans écrire les algorithmes, décrivez en français comment faire pour :
- vérifier que le nombre de champs est bien 3 ?
- extraire chacun des champs ?
- vérifier le nom de ville ?
- vérifier les coordonnées ?

Question 2 [facultatif]
Écrivez une fonction permettant d'extraire chacun des champs.

4.3 Mail

Question 1

Proposer un algorithme permettant de vérifier qu'une adresse mail est valide. Dans cet exercice nous considèrerons qu'une adresse mail est valide si :

- elle est composée de 3 parties : *partie1*`@`*partie2*`.`*partie3*
- si *partie1* et *partie2* ont entre 3 et 20 caractères
- si *partie3* a entre 2 et 7 caractères

Pour cet exercice on considèrera que tous les caractères existants sont autorisés dans *partie1* (excepté @), *partie2* (excepté .) et *partie3*. La norme est bien sûr plus complexe, voir notamment la RFC 5322. On note `decouper` la fonction "split" définie dans l'exercice précédent.

4.4 Palindromes

Question 1

Un palindrome est un mot (ou un texte) dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche, comme dans les phrases :

> « Ésope reste ici et se repose »

ou encore

> « La mariée ira mal »

à un accent près, et en ignorant les espaces.
Ecrivez une fonction prenant une Chaine en paramètre qui retourne vrai si elle est un palindrome et faux sinon.

4.5 Anagrammes [facultatif]

Une anagramme (le mot est féminin) est une construction fondée sur une figure de style qui inverse ou permute les lettres d'un mot ou d'un groupe de mots pour en extraire un sens ou un mot nouveau.

Ainsi les mots suivants sont tous des anagrammes entre eux :

Marion - manoir - minora - Romain - marino - romina - romani - mirano - mornai - normai

ainsi que :

ironique - onirique

ou encore :

chien - niche

Question 1

Écrire un algorithme permettant de tester si deux mots passés en paramètre sont des anagrammes.

Semaine 11

5 Fichiers : l'accès séquentiel

Le type Fichier. Pour cette séance de TD, les fichiers séquentiels sont des fichiers typés (par exemple, fichier d'entiers). Ainsi, on définit un type `Fichier` qui sera paramétré par le type de son contenu. Par exemple :

```
Var: f1 : Fichier d'entiers
     f2 : Fichier de Chaines
```

Nous ne nous soucions pas de la liaison entre la variable fichier manipulée par l'algorithme (ici, `f1` et `f2`) et les fichiers physiques contenus dans les répertoires. Cet aspect sera abordé avec la programmation `Java`.

Primitives. Les fichiers permettent de faire des entrées/sorties (*lectures* et *écritures*). Mais avant et après ces manipulations il faut prendre soin de les *ouvrir* dans un certain *mode d'accès* (`lecture, écriture, rajout`), et à la fin il faudra aussi les *fermer* en utilisant ces méthodes qui s'appliquent au type `Fichier` :

```
ouvrirFichier(modeAccès)
fermerFichier()
lireFichier() // renvoie un élément
écrireFichier(élément)
fin() // renvoie un booléen
```

Pour la lecture et l'écriture, le type du paramètre `élément` dépend bien sûr du type du fichier (dans un fichier d'entiers, on lit des entiers, etc). La primitive `fin()` permet de tester si la fin de fichier a été atteinte lors de la dernière opération de lecture (auquel cas cette opération n'a pas lu de nouvelle valeur).

Quelques particularités des fichiers.

accès séquentiel : les fichiers ne peuvent être lus que du début à la fin, c'est-à-dire qu'on ne peut pas accéder au 3ème élément d'un fichier sans avoir accédé aux 2 premiers, par exemple. C'est ce qu'on nomme *accès séquentiel*, à la différence des tableaux (*accès direct*) par exemple.

liaison logique - physique : en ExAlgo on ne se soucie pas de la couche physique (fichiers stockés en mémoire). En programmation, cela crée des cas particuliers à gérer : fichier absent, droits insuffisants, espace disque saturé, etc. Nous verrons en TP que `Java` utilise des *exceptions* pour gérer ces cas, et qu'il est très important de les traiter.

5.1 Parcours du contenu d'un fichier [facultatif]

Question 1
Écrire un algo permettant d'afficher à l'écran les *entiers pairs* contenus dans un fichier d'entiers.

5.2 Recherche (séquentielle bien sûr) dans un fichier

Question 1
Écrire un algorithme de recherche d'un élément dans un fichier d'entiers non trié.

Question 2 [facultatif]
Même question dans le cas trié.

5.3 Ajout d'un élément dans un fichier

Question 1
Écrire un algorithme permettant d'ajouter un entier dans un fichier d'entiers non trié.

Question 2
Même question dans le cas d'un fichier trié.

5.4 Suppression d'un élément dans un fichier

Question 1
Écrire un algorithme permettant de supprimer la première occurrence d'un entier dans un fichier d'entiers. Considérer le cas d'un fichier non trié et celui d'un fichier trié.

5.5 Fusion de deux fichiers triés

Question 1
Écrire un algorithme qui prend en paramètres deux fichiers d'entiers triés et les fusionne dans un troisième fichier trié aussi passé en paramètre.

Semaine 12

6 Récursivité

Un algorithme *récursif* est un algorithme qui s'invoque lui-même.

Illustration. Construire la représentation en numération binaire d'une valeur (entier positif en décimal). Exemple, la valeur 19 est représentée par "10011".

On remarque que :

— la représentation de 19, c'est le chiffre "1" (parce que 19 est impair) précédé par la représentation de $19/2 = 9$
— la représentation de 9, c'est "1" précédé par la représentation de $9/2 = 4$
— la représentation de 4, c'est "0" précédé par la représentation de $4/2 = 2$
— la représentation de 2, c'est "0" précédé par la représentation de $2/2 = 1$

Les cas de base sont 0 et 1 qui sont représentés par un seul chiffre.

On peut donc écrire la fonction `repBinaire` comme ceci :

```
Fonction repBinaire (E n : entier) : Chaine
Début
    Selon Que
        n = 0
        └ Retourner "0"
        n = 1
        └ Retourner "1"
       Sinon :
        └ Retourner repBinaire(n/2).concat(Chaine(n modulo 2))
```

Cet algorithme a une structure classique d'algorithme récursif :

— on trouve bien un *appel à l'algorithme lui-même*,
— les arguments de cet appel sont *plus petits* que les paramètres (ici, on passe de n à $n/2$),
— ils décroissent donc d'un appel à l'autre, jusqu'à arriver à un (ou plusieurs) cas de base. Sinon on obtiendrait une suite infinie d'appels : l'algorithme ne terminerait pas.
— on trouve donc un (ou plusieurs) *cas de base*, c'est-à-dire des valeurs d'arguments traités sans appel récursif (ici, 0 et 1),
— on s'autorise des instructions `Retourner` dans le corps du programme, étant donné cette séparation par cas.

6.1 Lecture d'un algorithme récursif

Considérons l'algorithme suivant :

```
Fonction somme (E n : entier) : entier
Début
    Si n = 0 Alors
     └ Retourner 0
    Sinon
     └ Retourner n + somme ( n − 1 )
```

Question 1
Que vaut `somme(4)` ? Écrivez la trace.

Question 2
Que renvoie cet algorithme ?

Question 3
Que renvoie `somme(-1)` ? Écrivez la trace.

6.2 Somme des entiers pairs

Question 1
Écrire un algorithme récursif qui étant donné n renvoie la somme des entiers pairs compris entre 0 et n.

6.3 Ping-Pong

Soient les deux actions suivantes :

```
Action ping(E n : entier)
Début
    Si n = 0 Alors
        Afficher ("Point Ping")
    Sinon
        pong( n − 1 )
        Afficher("Ping ")
```

```
Action pong(E n : entier)
Début
    Si n = 0 Alors
        Afficher ("Point Pong")
    Sinon
        ping( n − 1 )
        Afficher("Pong ")
```

Question 1
Expliquez et donnez la trace pour différents appels de `ping` pour différentes valeurs.

6.4 Système de fichiers

Un système de fichiers est organisé de manière arborescente : chaque répertoire peut contenir des fichiers et des répertoires. Dans cet exercice nous considérons qu'un répertoire n'est pas un fichier.

On dispose des fonctions `sousRepertoires` (resp. `fichiers`) pour récupérer les noms des répertoires (resp. fichiers) contenus dans un répertoire (cette fois-ci on ne compte pas les fichiers dans les sous-répertoires).

```
Fonction sousRepertoires(E repertoire : Chaine) : tableau de Chaines
Fonction fichiers(E repertoire : Chaine) : tableau de Chaines
```

Question 1
Écrire une fonction `nbFichiers` qui prend en paramètre un répertoire, et renvoie le nombre de fichiers qu'il contient, y compris dans les répertoires qu'il contient.

Question 2
Écrire une fonction `contientFichier` prenant en paramètre un répertoire et un nom de fichier, et qui renvoie `vrai` si le répertoire (ou l'un des répertoires qu'il contient) contient un fichier avec ce nom, et `faux` sinon.

Question 3 [facultatif]
Écrire une fonction `contientSsRepDeMemeNom` qui prend en paramètre un répertoire, et renvoie `vrai` si, parmi tous les répertoires (directs ou non) qu'il contient, deux ont le même nom, et `faux sinon`.

6.5 Fonction d'Ackerman [facultatif]

La fonction d'Ackerman $A : (m, n) \to A(m, n)$ est définie sur $N \times N$ par :

Si $m = 0$, alors $A(0, n) = n + 1$,
sinon, si $n = 0$, $A(m, 0) = A(m - 1, 1)$,
sinon $A(m, n) = A(m - 1, A(m, n - 1))$.

Question 1
Écrire un algorithme récursif calculant $A(m, n)$.

Question 2
Calculez $A(2, 2)$ en donnant la trace.

Semaine 13

---

7 Tris

Durant cette séance vous devez avoir compris le tri par sélection et le tri à bulle :
— savoir dérouler les deux algorithmes avec un tableau d'entier de taille 5.
— savoir ré-écrire les deux algorithmes.

7.1 Tri par sélection

Cf fiche 1.

7.2 Tri à bulle

Cf fiche 2.

Semaine 14

7.3 Tri par insertion

Le tri par insertion repose sur le principe suivant :
— A l'étape i (en comptant à partir de 0), les éléments de 0 à $i-1$ sont dans l'ordre croissant.
— On cherche alors à insérer le i-ème élément du tableau parmi ces éléments triés,
— pour cela, on décale ceux qui sont plus grands que lui.

Question 1
Faire tourner cet algorithme sur le tableau 5 3 7 4 2.

Question 2
Écrire l'algorithme de ce tri.

Question 3
Sur un tableau de taille n, combien de comparaisons ont lieu ?

Question 4
Combien d'échanges ont lieu ?

Question 5
Quel est le pire des cas ? Donner un exemple.

Question 6
Comment peut-on facilement diminuer le nombre de comparaisons ?

7.4 Tri rapide (quicksort)

Le tri rapide est le plus efficace en moyenne, parmi ceux présentés ici, malgré n^2 comparaisons dans le pire des cas.

Le principe du tri rapide consiste à choisir un élément "pivot", et à placer à sa gauche tous les éléments plus petits que lui, et à sa droite tous les éléments plus grands. Puis on fait de même pour le sous-tableau de gauche et le sous-tableau de droite, récursivement : c'est le principe "diviser pour régner". Pour simplifier, on choisira toujours comme pivot l'élément se trouvant au milieu du tableau à trier.

```
Action triRapide(ES tab : tableau d'entiers, E debut : entier, E fin : entier)
Var: pivot : entier
Début
    Si fin > debut Alors
        separer(tab, debut, fin, pivot)
        triRapide(tab, debut, pivot-1)
        triRapide(tab, pivot+1, fin)
```

Les entiers *debut* et *fin* déterminent entre quels indices doit avoir lieu le tri. Il suffit donc d'appeler *triRapide(tab, 0, taille-1)* pour trier tout le tableau.

L'action **separer** répartit les éléments du tableau en fonction du pivot. Le pivot est l'élément au milieu du tableau lors de l'appel. Les éléments qui lui sont inférieurs sont placés à sa gauche, les éléments

supérieurs à sa droite. Le pivot peut donc bouger : l'action **separer** renvoie son nouvel indice.

```
Action separer(ES tab : tableau d'entiers, E debut : entier, E fin : entier, S separation : entier)
Var: i : entier
Début
    echanger(tab, (debut+fin)/2, fin)
    separation ← debut
    Pour i de debut à fin-1 Faire
        Si tab[i] < tab[fin] Alors
            echanger(tab, i, separation)
            separation ← separation+1
    echanger(tab, fin, separation)
```

Question 1
Appliquer le tri rapide sur le tableau 13 8 9 11 7 5. Détailler.

Question 2
Quel est le pire des cas ? Combien de comparaisons ont lieu dans ce cas ?

7.5 Tri fusion (mergesort)

Le tri fusion illustre le concept "diviser pour régner". Il consiste à diviser le tableau initial en deux, à les trier (récursivement), puis à fusionner les deux sous-tableaux triés. En voici la fonction principale.

```
Action triFusion(ES tab : tableau[taille] d'entiers)
Début
    triFusionPartiel(tab, 0, taille-1)
```

La fonction `triFusionPartiel` effectue le tri récursivement.

```
Action triFusionPartiel(ES tab : tableau d'entiers, E debut : entier, E fin : entier)
Var: milieu : entier
Début
    Si fin > debut Alors
        milieu = (debut+fin)/2
        triFusionPartiel(tab, debut, milieu)
        triFusionPartiel(tab, milieu+1, fin)
        fusion(tab, debut, milieu, fin)
```

La fonction `fusion` prend deux portions consécutives du tableau déjà triées, et les fusionne de manière triée. Elle passe par un tableau intermédiaire. Nous avons déjà vu la fusion de fichiers triés (Exercice 5.5). Ici le principe est le même, il faut juste gérer les indices, alors que dans les fichiers ça se fait tout seul

(curseur).

```
Action fusion(ES tab : tableau[k] d'entiers, E debut : entier, E milieu : entier, E fin : entier)
Var: tmp : tableau[k] d'entiers // de même taille que tab
posGauche, posDroite, posTmp : entier
Début
    posGauche ← debut
    posDroite ← milieu+1
    posTmp ← debut
    Tant Que posGauche ≤ milieu et posDroite ≤ fin Faire
        Si tab[posGauche] < tab[posDroite] Alors
            tmp[posTmp] ← tab[posGauche]
            posGauche ← posGauche + 1
        Sinon
            tmp[posTmp] ← tab[posDroite]
            posDroite ← posDroite + 1
        posTmp ← posTmp + 1

    // on est arrivé au bout d'un des deux sous-tableaux (gauche ou droite)
    // remarque : un seul des deux tant-que ci-dessous sera exécuté.

    Tant Que posGauche ≤ milieu Faire
        tmp[posTmp] ← tab[posGauche]
        posGauche ← posGauche + 1
        posTmp ← posTmp + 1

    Tant Que posDroite ≤ fin Faire
        tmp[posTmp] ← tab[posDroite]
        posDroite ← posDroite + 1
        posTmp ← posTmp + 1

    // on recopie tmp dans tab
    Pour i de debut à fin Faire
        tab[i] ← tmp[i]
```

Question 1
Faire tourner le tri fusion sur le tableau [5,3,7,4,1,2].

Semaine 15

8 Conteneurs

8.1 Agenda

On souhaite stocker des événements dans l'ordre croissant :
— Rendez-Vous DDE : 2018.01.10, 10h45
— DS Rattrapage Algo : 2018.01.14 : 06h00
— ...

Question 1
Comment stockerait-on un tel agenda en Java, en supposant que l'on dispose d'une classe `Evenement` ?

Question 2
Le tableau suivant énumère un certain nombre de besoins fonctionnels. Remplissez ce tableau en indiquant les solutions algorithmiques, et en précisant si cette structure est adaptée à chaque besoin.

Besoin algo	Solution langage naturel	SD adaptée (O/N)	Potentielle limitation
Importer un calendrier depuis un fichier			
Affichage du calendrier / semaine / journée (ordre essentiel)			
Recherche d'un événement à partir de sa description			
Annulation d'un rendez-vous			
Journée banalisée : on veut effacer les k événements d'un jour donné (intervalle de k indices)			
Ajout d'un événement			

Question 3
Quel bilan tirer de ce tableau ? Cette structure vous semble-t-elle adaptée à cette demande ?

Question 4
Idéalement, quelle genre de structure répondrait mieux à ces besoins fonctionnels ?

8.2 Salles d'enseignement du département

Dans cet exercice on souhaite pouvoir gérer les salles d'enseignement du département Info : leur numéro (entre 1 et 319), et leur description (Machine, TD, Anglais, Expression, AS, Screen).

Nous allons considérer 4 structures de données possibles pour résoudre ce problème :

Version 1 : tableau de descriptions, indice case = numéro de salle, chaine "" si pas de salle.	`final int MAX = 320;` `String[] tabSalles = new String[MAX];`

Version 2 : tableau d'objets (numéro, description)	`final int MAX = 20;`
	`class Salle {`
	`  int numero;`
	`  String description;`
	`}`
	`Salle[] tabSalles = new Salles[MAX];`

Version 3 : idem Version 2, mais trié selon les numéros	
Version 4 : tableau extensible, trié par numéro de salle.	`ArrayList<Salle> tabSalles =` `  new ArrayList<>();`

Question 1
Remplir le tableau suivant pour chacune des 4 versions proposées.

Besoin algo	Solution langage naturel	SD adaptée (O/N)	Potentielle limitation
Afficher liste des salles sous forme "num : desc"	V1 : V2 : V3 : V4 :	V1 : V2 : V3 : V4 :	V1 : V2 : V3 : V4 :
Idem mais par ordre croissant des numéros	V1 : V2 : V3 : V4 :	V1 : V2 : V3 : V4 :	V1 : V2 : V3 : V4 :
Ajouter une salle (num, desc)	V1 : V2 : V3 : V4 :	V1 : V2 : V3 : V4 :	V1 : V2 : V3 : V4 :
Vérifier si une salle de num donné est déjà présente	V1 : V2 : V3 : V4 :	V1 : V2 : V3 : V4 :	V1 : V2 : V3 : V4 :

Question 2
Quels problèmes sont résolus par les tableaux extensibles (par rapport aux tableaux simples) ? Par contre, quels problèmes ne sont pas résolus ?

Question 3
Quel type de structure permet de résoudre ces problèmes ?

Semaine 16

Sur machine : utilisation des conteneurs.

A Bonnes pratiques

Vérifier que :

1. toute variable utilisée est *déclarée* dans **Var**,
2. toute variable déclarée dans **Var** est *utilisée* (sinon, la supprimer),
3. toute variable est *initialisée* avant son utilisation,
4. les *accès* **E**, **S** *et* **ES** des paramètres sont cohérents avec le programme,
5. les fonctions ont un *type de retour*, pas les actions,
6. les fonctions n'ont que des paramètres en entrée (**E**),
7. une fonction *retourne* une valeur, dans tous les cas,
8. pas d'*instructions* `Retourner` *dans les boucles* au S1,
9. les *noms* (variables, fonctions, etc) sont *explicites* et en *camelCase*,
10. les blocs sont bien *délimités et indentés*,
11. les boucles se *terminent* dans tous les cas,
12. les *cas limites* sont bien gérés (tableaux vides, etc),
13. pas de comparaison aux valeurs `vrai` et `faux`
    (exemple : `trouvé = faux` à remplacer par `non trouvé`).